The Journal of Open Source Software

t8code - modular adaptive mesh refinement in the exascale era

Johannes Holke [1], Johannes Markert [1*¶], David Knapp [1*], Lukas Dreyer [1*], Sandro Elsweijer [1*], Niklas Böing [1*], Chiara Hergl [1*], Prasanna Ponnusamy [1*], Jakob Fussbroich [1*], Tabea Leistikow [1*], Florian Becker [1*], Ioannis Lilikakis [1*], and Carsten Burstedde [2]

1 German Aerospace Center (DLR), Institute of Software Technology, Department High-Performance Computing, Cologne, Germany **2** Rheinische Friedrich-Wilhelms-Universität Bonn, Institute for Numerical Simulations and Hausdorff Center for Mathematics, Germany ¶ Corresponding author * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

In this paper, we present our scalable dynamic adaptive mesh refinement (AMR) library t8code, which was officially released in 2022 (Holke et al., 2022). t8code is written in C/C++, open source, and readily available at dlr-amr.github.io/t8code. It is developed and maintained at the Institute for Software Technology of the German Aerospace Center (DLR). The software library provides fast and memory efficient parallel algorithms for dynamic AMR to handle tasks such as mesh adaptation, load-balancing, ghost computation, feature search and more. t8code can manage meshes with over one trillion mesh elements (Holke et al., 2021) and scales up to one million parallel processes (Holke, 2018). It is intended to be used as mesh management backend in scientific and engineering simulation codes paving the way towards high-performance applications of the upcoming exascale era.

Statement of Need

Adaptive Mesh Refinement has been established as a successful approach for scientific and engineering simulations over the past decades (Babuvška & Rheinboldt, 1978; Bangerth et al., 2007; Dörfler, 1996; Teunissen & Keppens, 2019). By modifying the mesh resolution locally according to problem specific indicators, the computational power is efficiently concentrated where needed and the overall memory usage is reduced by orders of magnitude. However, managing adaptive meshes and associated data is a very challenging task, especially for parallel codes. Implementing fast and scalable AMR routines generally leads to a large development overhead motivating the need for external mesh management libraries like t8code.

Currently, t8code's AMR routines support a wide range of element types: vertices, lines, quadrilaterals, triangles, hexahedra, tetrahedra, prisms, and pyramids. Additionally, implementation of other refinement patterns and element shapes is possible. See Figure 1 for an examplary adapted mesh managed by t8code for visualizing the temperature profile of a convection simulation of a model planet's mantle (source: Institute of Planetary Research, DLR). The original, uniform mesh consists of over 158 million cells allocating 6.818 GB of memory. By applying AMR to the data the memory usuage could be reduced to 20% with an compression error of less then 1%. The error meassure was chosen to be the norm of the variance between refinement/coarsening steps. That is, starting from the uniform mesh at highest refinement level ($l = 8$), the mesh was successively coarsened till the disagreement from the original data reached 1%. It should be noted that t8code's primary objective is to provide flexible adaptive mesh management. The layout of the data inside an element and its interpretation regarding, for example, when and how to refine/coarsen is up to the application linking against t8code.

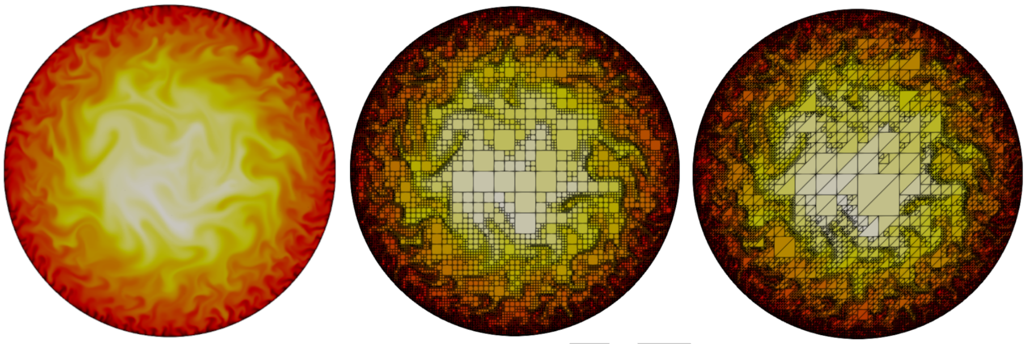

Figure 1: Visulization of a planetary mantle convection simulation (source: Institute of Planetary Research, DLR). Shown is the 2D slice of the temperatur profile. Left: original uniform data. The highlighting of the grid lines was omitted for visual clarity. Middle: adapted mesh with quad elements. Right: adapted mesh with triangle elements. The original data living on a uniform quad mesh was first transfered to a triangle mesh and adapted afterwards. This shows the versatility of t8code regarding to the choice of mesh elements.

Fundamental Concepts

`t8code` is based on the forest-of-trees approach. Starting point for the usage of `t8code` is an unstructured conformal input mesh, which we denote a coarse mesh. This coarse mesh describes the geometry of the computational domain. Each of the coarse mesh cells is then viewed as the root of a refinement tree. These trees are refined recursively in a structured pattern, resulting in a collection of trees, which we call a forest. `t8code` stores only a minimal amount of information about the finest elements of the mesh - he leaves of the trees - in order to reconstruct the whole forest.

By enumerating the leaves in a recursive refinement pattern we obtain a space-filling curve (SFC) logic. Via these SFCs, all elements in a refinement tree are assigned an integer-based index and are stored in linear order. Element coordinates or element neighbors do not need to be stored explicitly but can be reconstructed from the SFC index. Fast bitwise SFC operations ensure optimal runtimes and diminish the need for memory lookups. Moreover, the SFC is used to distribute the forest mesh across multiple processes, so that each process only stores a unique portion of the SFC. See Figure 2.

While being successfully applied to quadrilateral and hexahedral meshes (Burstedde et al., 2011; Weinzierl, 2019), these SFC techniques are extended by `t8code` in a modular fashion, such that arbitrary element shapes are supported. We achieve this modularity through a novel decoupling approach that separates high-level (mesh global) algorithms from low-level (element local) implementations. All high-level algorithms can be applied to different implementations of element shapes and refinement patterns. A mix of different element shapes in the same mesh is also supported.

Mesh adapation as it is done in t8code leads to hanging nodes. Numerical methods have to speficially handle these non-conforming interfaces. Finite-Volume schemes or Discontinuous Galerkin methods naturally treat this problem via so-called mortar methods. In the future, it is planned to also support hanging nodes resolving routines by inserting transition elements conformally connecting elements at different refinement levels.

The Journal of Open Source Software

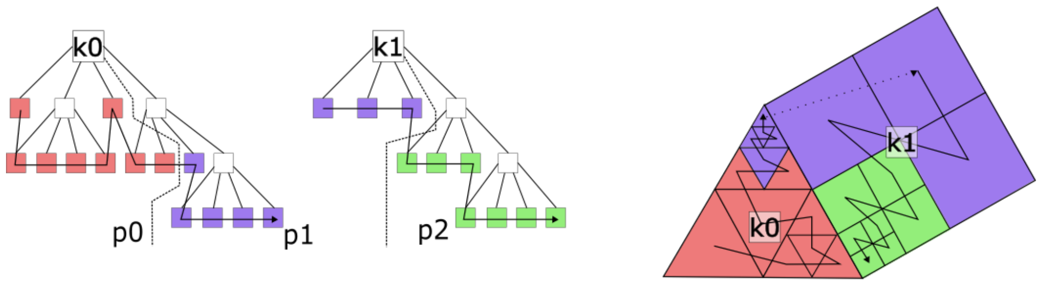

Figure 2: Left: Exemplary t8code forest mesh consisting of two trees (k0, k1) distributed over three parallel processes p0 to p2. The SFC is represented by a black curve tracing only the finest elements (leaves) of each tree. Right: Sketch of the associated mixed shape (a triangle and a quad) mesh refined up to level three.

Performance

`t8code` supports distributed coarse meshes of arbitrary size and complexity, which we tested for up to 370 million coarse mesh cells (Burstedde & Holke, 2017). Moreover, we conducted various performance studies on the JUQUEEN and the JUWELS supercomputers at the Jülich Supercomputing Center. In Table 1, (Holke et al., 2021) we show that `t8code`'s ghost routine is exceptionally fast with proper scaling of up to 1.1 trillion mesh elements. Computing ghost layers around parallel domains is usually the most expensive of all mesh operation. Furthermore, in a prototype code (Dreyer, 2021) implementing a high-order discontinuous Galerkin method (DG) for advection-diffusion equations on dynamically adaptive hexahedral meshes we can report of a 12 times speed-up compared to non-AMR meshes with only an overall 15% runtime contribution of `t8code`. In fig. Figure 3 we compare the runtimes over number of processes of the DG solver and the summed mesh operations done by t8code which are ghost computation, ghost data exchange, partitioning (load balancing), refinement and coarsening as well as balancing ensuring only a difference of one refinement level among element's face neighbors. Additionally, from the graph we see the weak scaling property of the application, i.e. the runtime halves when doubling the number of processes.

# Process	# Elements	# Elem. / process	Ghost
49,152	1,099,511,627,776	22,369,621	2.08 s
98,304	1,099,511,627,776	11,184,811	1.43 s

Table 1: Runtimes on JUQUEEN for the ghost layer computation for a distributed mesh consisting of 1.1 trillion elements.

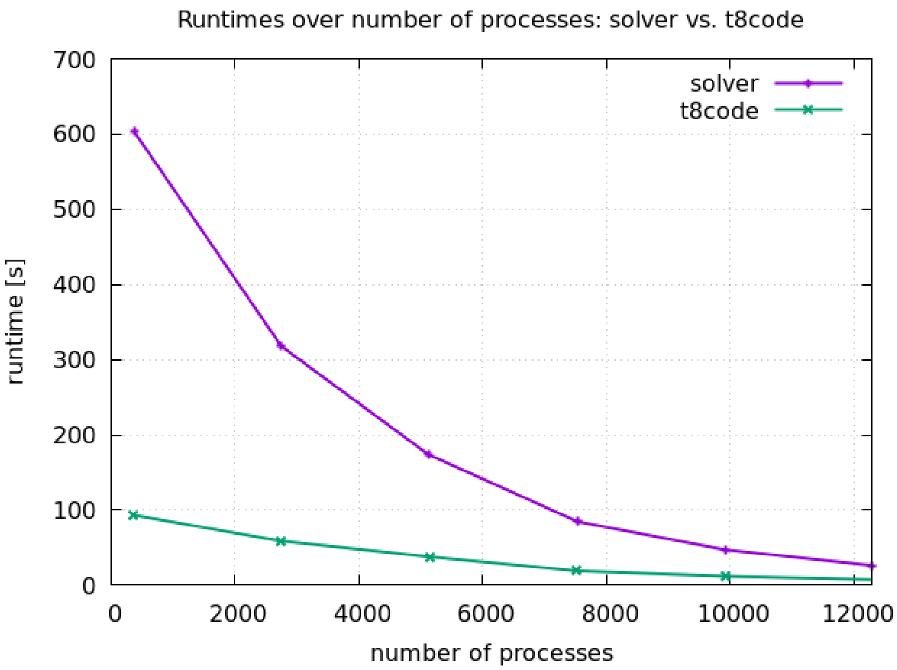

Figure 3: Runtimes on JUQUEEN of the solver and summed mesh operations of our DG prototype code coupled with `t8code`. Mesh operations are ghost computation, ghost data exchange, partitioning (load balancing), refinement and coarsening as well as balancing. `t8code` only takes around 15% of the overall runtime. Additionally, we see the weak scaling property of the application, i.e. the runtime halves when doubling the number of processes.

Research Projects

Even though `t8code` is a newcomer to the market, it is already in use as the mesh management backend in various research projects, most notably in the earth system modeling (ESM) community. In the ADAPTEX project `t8code` is integrated with the Trixi framework (Schlottke-Lakemper et al., 2020) - a modern computational fluid dynamics code written in Julia. Over the next years several ESM applications are planned to couple to this combination, including MESSy, MPTrac, and SERGHEI. Moreover, `t8code` also plays an important role in several DLR funded research projects, e.g., VisPlore (massive data visualization), HYTAZER (hydrogen tank certification), and Greenstars (additive rocket engine manufacturing).

Further Information

For further information beyond this short note and also for code examples, we refer to our Documentation and Wiki reachable via our homepage dlr-amr.github.io/t8code and our technical publications on `t8code` (Becker, 2021; Burstedde & Holke, 2016, 2017; Dreyer, 2021; Elsweijer, 2021, 2022; Fußbroich, 2023; Holke, 2018; Holke et al., 2021, 2022; Knapp, 2020; Lilikakis, 2022).

Acknowledgements

Johannes Holke thanks the Bonn International School Graduate School of Mathematics (BIGS) for funding the initial development of `t8code`. Further development work was funded by the German Research Foundation as part of project 467255783, the European Union via

NextGenerationEU and the German Federal Ministry of Research and Education (BMBF) as part of the ADAPTEX and PADME-AM projects. Development work was performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received funding from the Helmholtz Association of German Research Centres. The development team of `t8code` thanks the Institute for Software Technology and the German Aerospace Center (DLR).

The authors state that there are no conflicts of interest.

References

Babuvška, I., & Rheinboldt, W. C. (1978). Error estimates for adaptive finite element computations. *SIAM Journal on Numerical Analysis*, *15*(4), 736–754. https://doi.org/10.1137/0715049

Bangerth, W., Hartmann, R., & Kanschat, G. (2007). Deal.II—a general-purpose object-oriented finite element library. *ACM Trans. Math. Softw.*, *33*(4), 24–es. https://doi.org/10.1145/1268776.1268779

Becker, F. (2021). *Removing hanging faces from tree-based adaptive meshes for numerical simulations* [Master's thesis]. Universität zu Köln.

Burstedde, C., & Holke, J. (2016). A tetrahedral space-filling curve for nonconforming adaptive meshes. *SIAM Journal on Scientific Computing*, *38*, C471–C503. https://doi.org/10.1137/15M1040049

Burstedde, C., & Holke, J. (2017). Coarse Mesh Partitioning for Tree-Based AMR. *SIAM Journal on Scientific Computing*, *Vol. 39*, C364–C392. https://doi.org/10.1137/16M1103518

Burstedde, C., Wilcox, L. C., & Ghattas, O. (2011). p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SIAM Journal on Scientific Computing*, *33*(3), 1103–1133. https://doi.org/10.1137/100791634

Dörfler, W. (1996). A convergent adaptive algorithm for poisson's equation. *SIAM Journal on Numerical Analysis*, *33*(3), 1106–1124. https://doi.org/10.1137/0733054

Dreyer, L. (2021). *The local discontinuous galerkin method for the advection-diffusion equation on adaptive meshes* [Master's thesis, Rheinische Friedrich-Wilhems-Universität Bonn]. https://elib.dlr.de/143969/

Elsweijer, S. (2021). *Curved Domain Adaptive Mesh Refinement with Hexahedra*. Hochschule Bonn-Rhein-Sieg. https://elib.dlr.de/143537/

Elsweijer, S. (2022). *Evaluation and generic application scenarios for curved hexahedral adaptive mesh refinement* [Master's thesis, Hochschule Bonn-Rhein-Sieg]. https://doi.org/10.13140/RG.2.2.34714.11203

Fußbroich, J. (2023). *Towards high-order, hybrid adaptive mesh refinement: Implementation and evaluation of curved unstructured mesh elements* [Master's thesis, Technische Hochschule Köln]. https://elib.dlr.de/200442/

Holke, J. (2018). *Scalable algorithms for parallel tree-based adaptive mesh refinement with general element types* [PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn]. https://doi.org/20.500.11811/7661

Holke, J., Burstedde, C., Knapp, D., Dreyer, L., Elsweijer, S., Uenlue, V., Markert, J., Lilikakis, I., & Boeing, N. (2022). *t8code* (Version 1.0.0). https://doi.org/10.5281/zenodo.7034838

Holke, J., Knapp, D., & Burstedde, C. (2021). An Optimized, Parallel Computation of the Ghost Layer for Adaptive Hybrid Forest Meshes. *SIAM Journal on Scientific Computing*, C359–C385. https://doi.org/10.1137/20M1383033

Knapp, D. (2020). *A space-filling curve for pyramidal adaptive mesh refinement* [Master's Thesis]. Rheinische Friedrich-Wilhelms-Universität Bonn.

Lilikakis, I. (2022). *Algorithms for tree-based adaptive meshes with incomplete trees* [Master's thesis, Universität zu Köln]. https://elib.dlr.de/191968/

Schlottke-Lakemper, M., Gassner, G. J., Ranocha, H., & Winters, A. R. (2020). *Trixi.jl: Adaptive high-order numerical simulations of hyperbolic PDEs in Julia*. https://github.com/trixi-framework/Trixi.jl. https://doi.org/10.5281/zenodo.3996439

Teunissen, J., & Keppens, R. (2019). A geometric multigrid library for quadtree/octree AMR grids coupled to MPI-AMRVAC. *Computer Physics Communications*, *245*, 106866. https://doi.org/10.1016/j.cpc.2019.106866

Weinzierl, T. (2019). The Peano Software-Parallel, Automaton-based, Dynamically Adaptive Grid Traversals. *ACM Transactions on Mathematical Software*, *45*(2), 1–41. https://doi.org/10.1145/3319797